*International Institute of Information Technology, Bangalore*



**MLOps Project Report**

**Project Title**: **End-to-End MLOps with Kubeflow**

**Under the Guidance of** : **Prof. Uttam Kumar**

**Pranav Kumar**

IMT2018020

*pranav.kumar@iiitb.ac.in*

Content

[1 Abstract 3](#_Toc122262637)

[2 Summary on MLOps 3](#_Toc122262638)

[2.1 Introduction 3](#_Toc122262639)

[2.2 Benefits of MLOps 4](#_Toc122262640)

[2.3 DevOps vs MLOps 4](#_Toc122262641)

[2.4 Tools Available 5](#_Toc122262642)

[2.4.1 MLflow 5](#_Toc122262643)

[2.4.2 TensorBoard 6](#_Toc122262644)

[3 Kubeflow 6](#_Toc122262645)

[3.1 Introduction 6](#_Toc122262646)

[3.2 Installation Guide 7](#_Toc122262647)

[3.3 Kubeflow Dashboard 7](#_Toc122262648)

[3.4 Kubeflow Pipelines 7](#_Toc122262649)

[3.4.1 Ways to build pipelines in Kubeflow 7](#_Toc122262650)

[3.4.2 Building a Kubeflow Pipeline 7](#_Toc122262651)

[4 Project Details 8](#_Toc122262652)

[4.1 Aim 8](#_Toc122262653)

[4.2 Objectives 8](#_Toc122262654)

[4.3 Building the ML Model 8](#_Toc122262655)

[4.4 Building the Kubeflow Pipeline 8](#_Toc122262656)

[4.4.1 Deploying Kubeflow 8](#_Toc122262657)

[4.4.2 Opening Jupyter Notebook Server 9](#_Toc122262658)

[4.4.3 Initiating the Kubeflow pipelines SDK 9](#_Toc122262659)

[4.4.4 Converting to Docker containers 9](#_Toc122262660)

[4.4.5 Defining the Kubeflow pipeline 9](#_Toc122262661)

[4.4.6 Creating a Persistent Volume 10](#_Toc122262662)

[4.4.7 Defining the pipeline components 10](#_Toc122262663)

[4.4.8 Running the pipeline 11](#_Toc122262664)

[5 Outcome 11](#_Toc122262665)

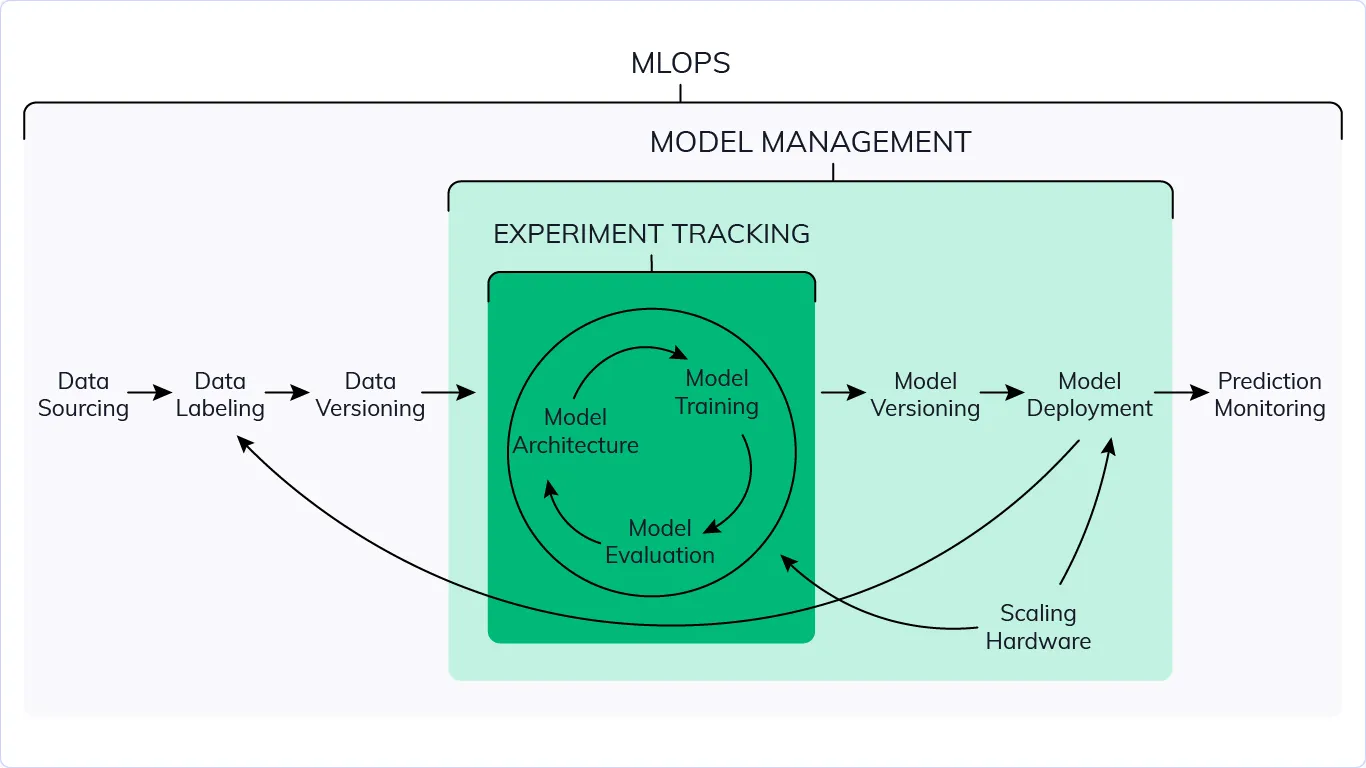
[6 References 12](#_Toc122262666)

# Abstract

# Summary on MLOps

## Introduction

MLOps which stands for Machine Learning Operations is an ML Engineering culture practice that aims at unifying ML system development (Dev) and ML system operations (Ops). MLOps is all the engineering pieces that come together and often help to deploy, run, and train AI models.



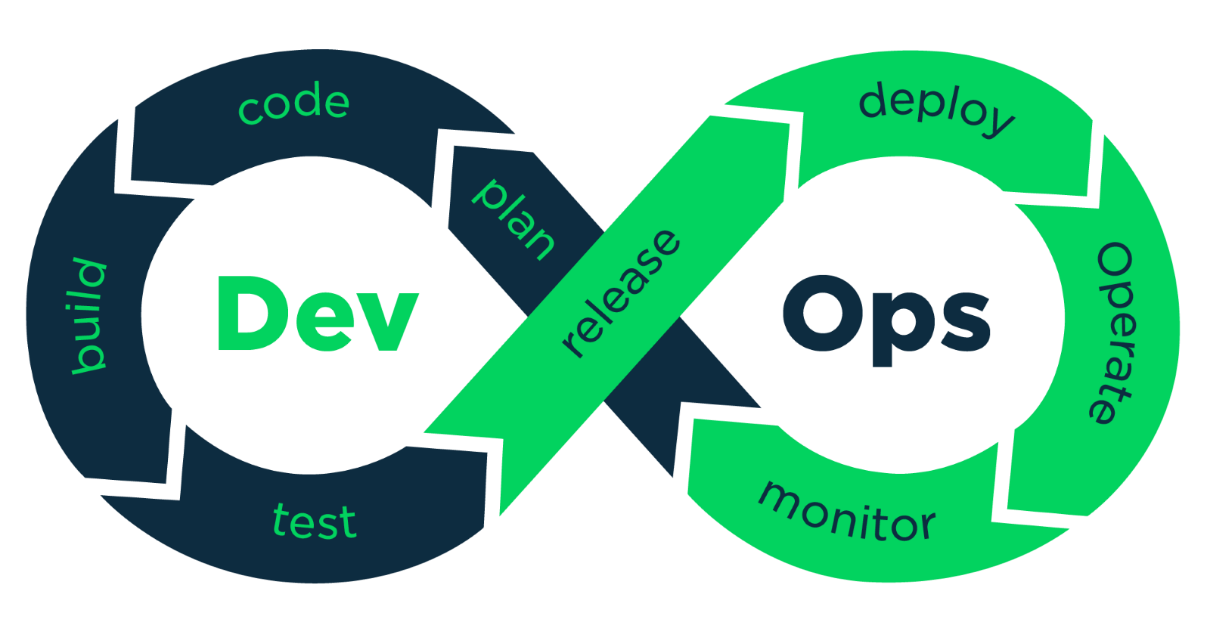
The ML system lifecycle involves several different teams of a data driven organization. The machine learning lifecycle consists of many complex components such as data ingest, data prep, model training, model tuning, model deployment, model monitoring, explain ability, and much more. It also requires collaboration and hand-offs across several teams, from Data Engineering to Data Science to ML Engineering.

## Benefits of MLOps

* **Productivity**
  + - Adopting the practice of MLOps increases the ***productivity*** within all processes within the Machine Learning Lifecycle of the model to be deployed.
    - MLOps, which stands for automated workflow of the ML Model from data collection to the model deployment.
    - It saves time for the all the teams part of the lifecycle and prevents ***human-induced*** ***errors***.
    - MLOps practices ***inspires collaboration*** between all the members of the tam and also enable businesses to standardize the entire ML Workflow.
* **Cost Reduction**
  + - Costs can be considerably reduced throughout the whole machine learning lifecycle with MLOps.
    - The automation mimnizes the need for ***manual labour*** to manage the machine learning models.
    - The more systematic approach reduces the ***time for error-detection*** to find the stage at which the error was caused in case of any.
* **Monitorability**
  + - MLOps enables businesses to monitor and ***get better insights*** about model performance systematically.
    - The machine learning models are continuously ***retrained*** on any event trigger or periodically to ensure the model provides the most accurate output.
    - Adopting the practice enables the businesses to give real-time status of the model and alert the relevant teams about any kind of model performance degrades or errors.
* **Reliability**
  + - Incoparating CI/CD pipelines from DevOPs into the machine learning processes, MLOps makes ML Pipelines more reliable.
    - Automated ML lifecycle reduces human error while providing businesses with accurate data and insight.
    - Model management procedures are streamlined by MLOps to allow accurate scalability.
* **Reproducibility**
  + - Automating ML workflow allows for reliability and predictability in the deployment of the machine learning model. This helps in becoming more productive by reducing the time to deploy models.
    - MLOps makes sure to save snapshots of various versions of data sets that were created or altered at different points in time.
    - Through MLOps the model is versioned with various hyperparameters and model kinds. Feature stores are created for various sorts of model characteristics.

## DevOps vs MLOps

DevOps is a culture and set of processes that brings together development and operations teams to finish software development. It enables businesses to develop and improve products at a faster rate than they could which use traditional software development methods.



## Tools Available

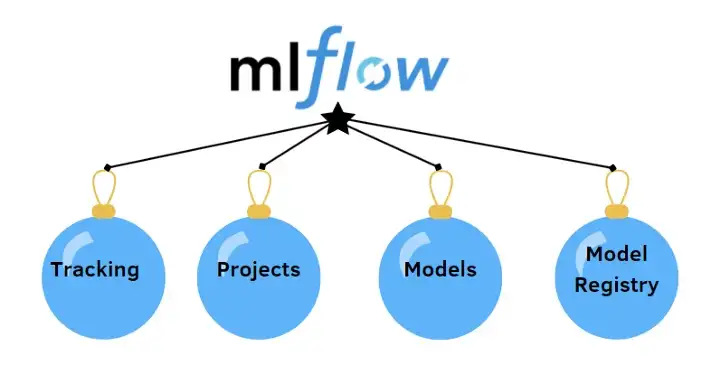
There are several platforms that are available for making and deploying the required pipelines.

### MLflow

MLflow is a framework that supports the machine learning lifecycle.  This means that it has components to monitor your model during training and running, ability to store models, load the model in production code and create a pipeline.

MLflow has four critical components:

* **Tracking:** allows to log or record model training sessions
* **Projects**:
* **Models**:
* **Model registry:**



### TensorBoard

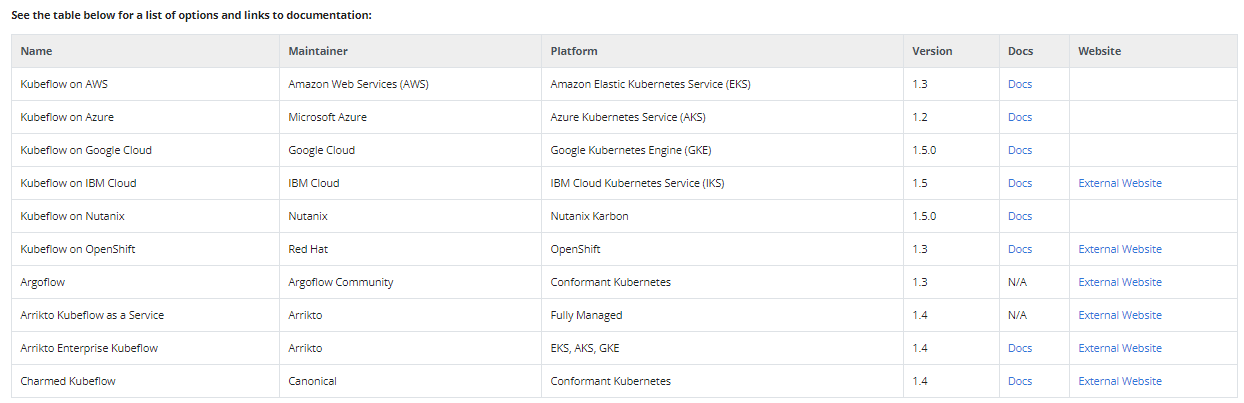
# Kubeflow

Kubeflow makes it easier for the deployment of machine learning projects on Kubernetes. Kubeflow pipelines are a great way to build portable and scalable machine learning workflows. They simplify orchestration of Machine Learning Pipelines at a larger scale. A general pipeline is a sequence of clearly defined steps in a ML workflow.

The pipeline must be defined the necessary parameter inputs and the inputs and outputs of each component. When a pipeline is executed Kubeflow launches a single or multiple Kubernetes Pods corresponding to the components defined in the pipeline. The pods start the docker container which executes the code written in the component.

## Introduction

## Installation Guide

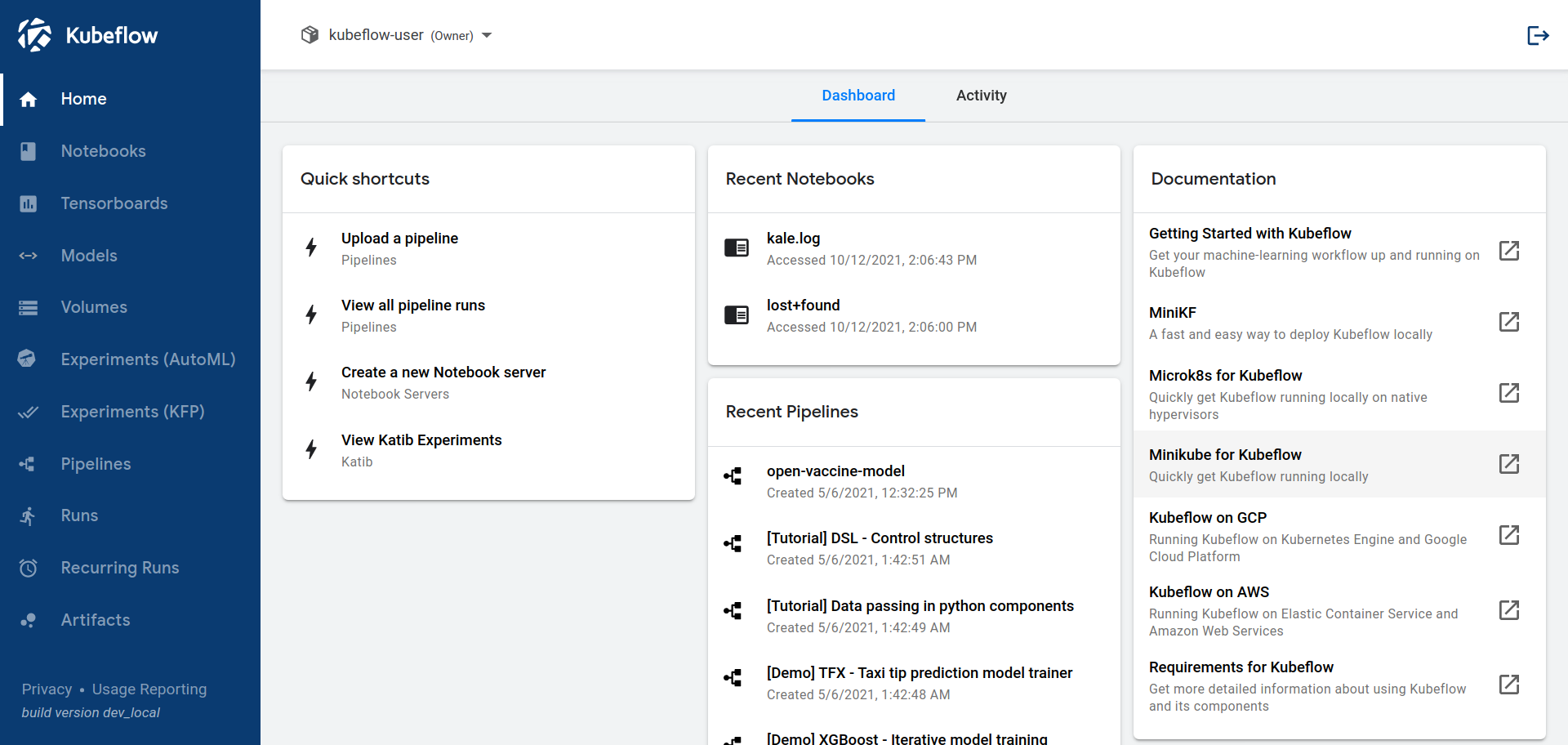
* There are two ways to install Kubeflow
  + - Installing it through manifest
    - Installing it with a packaged distribution. Packaged distributions are supported by respective maintainers. For instance Kubeflow on Azure is maintained by Microsoft.

## Kubeflow Components

Kubeflow components are logical blocks that together make up Kubeflow.A machine learning project will normally require one to use one or multiple components.

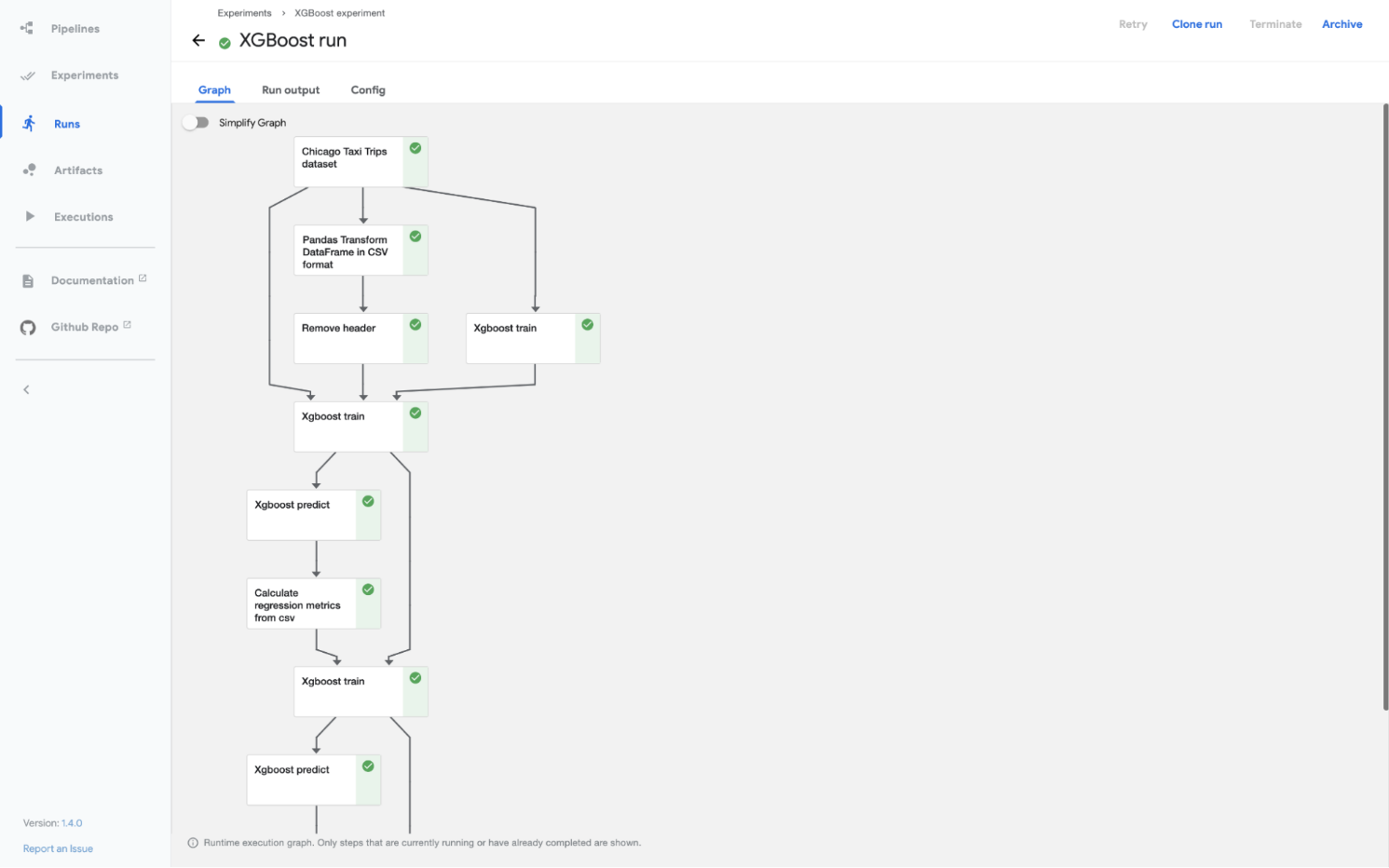
### Dashboard

* Once the Kubeflow package is deployed successfully the dashboard can be accessed from the browser using the input parameter URL given during deployment.
* The Central Dashboard provides quick access to other Kubeflow components. Some of the useful components are:
  + - **Notebooks : To manage Notebooks servers**
    - **Pipelines : To manage Kubeflow Pipelines**
    - **Experiments: To manage the Katib experiments**



### Pipelines

* Kubeflow Pipelines is a powerful component to allow users to build portable and scalable machine learning pipelines based on Docker Containers.
* Each step in a pipeline is a Docker container, hence portable and scalable. This also makes each step in the pipeline independent wooing one to reuse the pipeline components

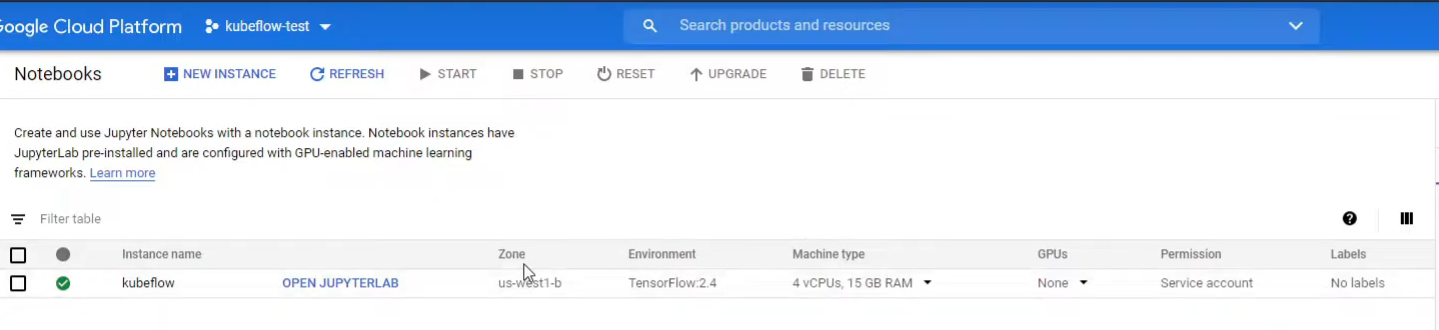


#### Building a Kubeflow Pipeline

* For this project/experiment we develop a Neural Net model to perform image classification on the Fashion MNIST dataset with TensorFlow and turn it into a Kubeflow pipeline.

### Notebooks

* Kubeflow comes with an integrated Jupyter notebook environment allowing the user to directly launch a notebook server.
* Notebooks enable users to perform quick tests, develop models and writing machine learning applications
* By running the notebooks locally the engineers are constrained by resources. Having the notebooks in the cluster makes it easy to run jobs where the resources can be dynamically scaled.

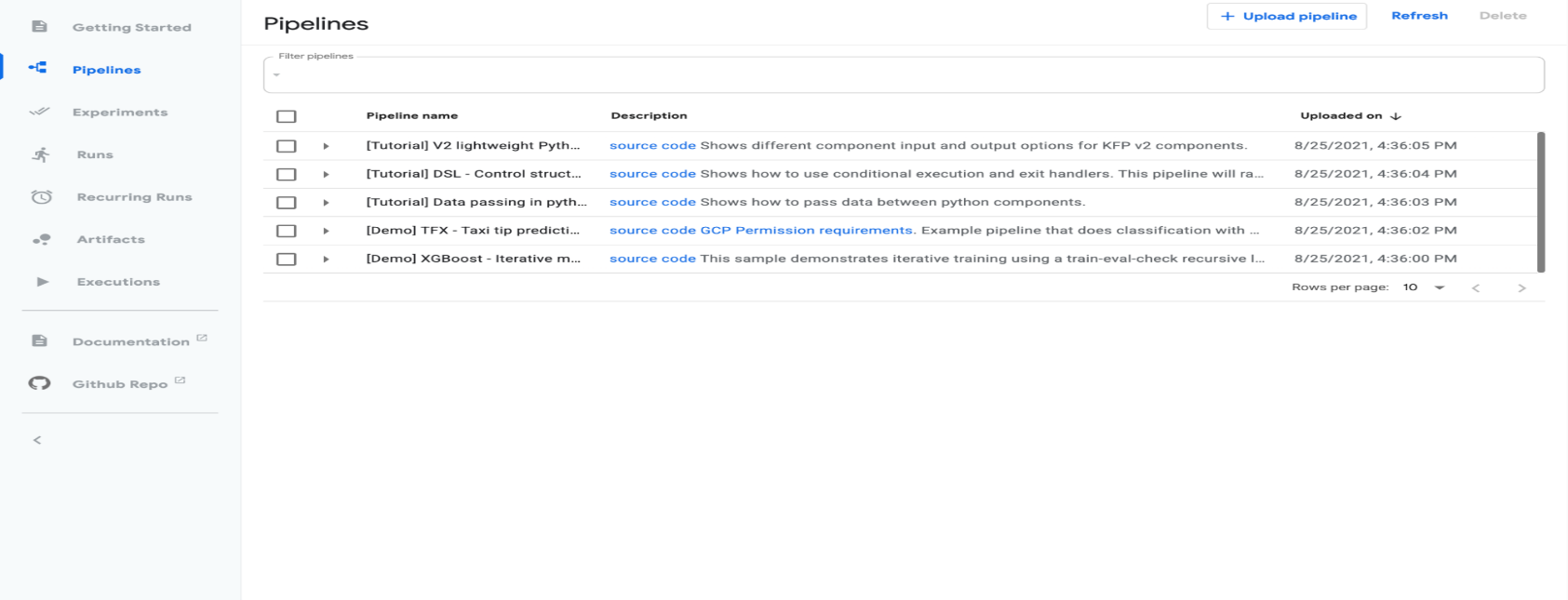


## Kubeflow Pipelines

* Since a general machine learning workflow can involve several steps, from preparing the data to model training to model evaluation and more , it makes it difficult to track each step in an ad-hoc manner.
* Model version tracking without a proper tooling is another challenge for data scientists to overcome.
* Kubeflow Pipelines let data scientists develop machine learning workflows using standards that are composable, shareable, and reproducible.
* Each component in a Kubeflow pipeline is generally a Docker container. A pipeline component is a self-contained code that performs a step in the ML workflow.
* Since, each component in a pipeline is a packaged Docker image so each step or component is independent of each other allowing to mix and match different frameworks in a ML workflow.

### Ways to build pipelines in Kubeflow

#### User Interface (UI)

* The Kubeflow Dashboard UI can directly be used to run pipelines or even upload pipelines from elsewhere. There are additional features that can be used sucxh as scheduling automated runs, view artifacts and output of the pipeline runs.

#### REST API

* Kubeflow provides REST APIs for continuous integration and deployment systems.

#### Python SDK

* Kubeflow provides Python libraries that the user can use to develop and execute ML pipelines.

# Project Details

## Aim

The aim of the project is to experiment Kubeflow to deploy a Machine Learning Model.

## Objectives

For the current project the Machine Learning Model used to deploy yhte pipeline is a model that solves the Fashion MNIST clothing classification problem used in computer vision and deep learning.

## Building the ML Model

### The Dataset

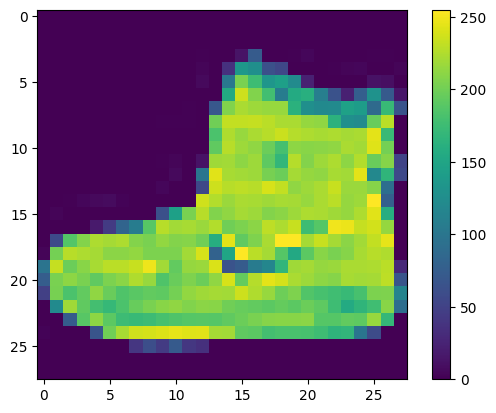
* For this project, a neural network model is trained to classify images of several pieces of clothing such as sneakers, shirts etc.
* *tf.keras*, a high-level API is used to build and train the models in TensorFlow.
* The dataset chosen for this project is the Fashion MNIST dataset, which contains over 70,000 grayscale images in 10 categories, each with a resolution of (28x28 pixels).
* Off the dataset 60,000 are used to r=train the network and the rest 10,000 to evaluate and test the model.

The mapping of all 0-9 integers to class labels is listed below.

* 0: T-shirt/top
* 1: Trouser
* 2: Pullover
* 3: Dress
* 4: Coat
* 5: Sandal
* 6: Shirt
* 7: Sneaker
* 8: Bag
* 9: Ankle boot
* The problem is more challengeing then the general MNIST and the top results are only achived through deep learning convolution neural networks with high classification accuracy on the hold out test dataset

### Data Preprocessing

* Before the network to be trained, the data has to be preprocessed accordingly. When inspecting the first image, we can notice htat the pixel values fall in the range of 0 to 255
* We normalize the values to range between 0 to 1. To normalize, we divide the values by 255.
* The training and the testing dataset has to be pre-processed the same way.



### Building the model

* The layers of the model must first be configured before the model can be assembled to create the neural network.
* The layer is a neural network's fundamental building unit. The data that is fed into layers is used to extract representations.
* Chaining collectively simple layers makes up the majority of deep learning.
* *tf.keras.layers.Dense* layer used in the project has parameters that are learned during training.

model = tf.keras.Sequential([  
    tf.keras.layers.Flatten(input\_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dense(10)  
])

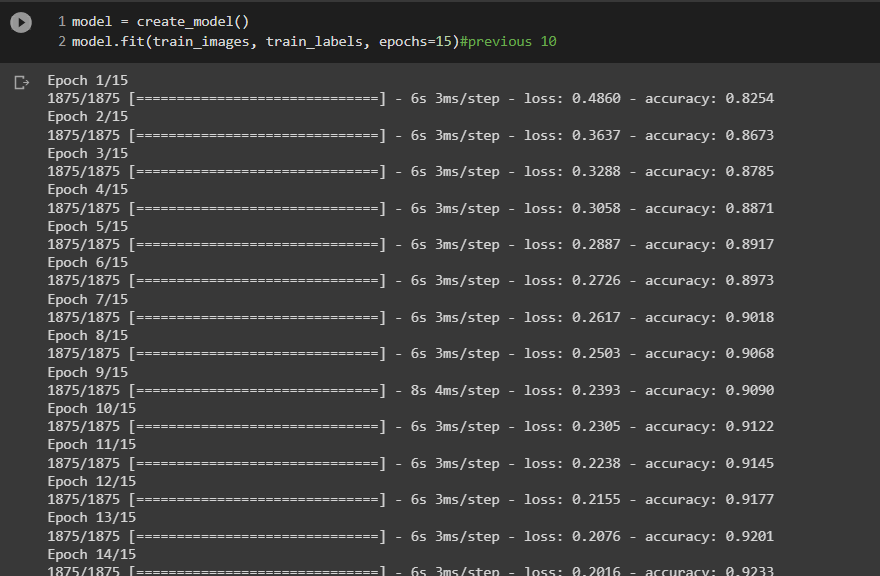
* The first layer *tf.keras.layers.Flatten* transform the 2D Image array (28 x 28 pixels) into a 1D Array of dimension ( 28 \* 28 = 784 pixels ) . This layer only reformats the data.
* The sequence of two *tf.keras.layers.Dense* layers are densely fully connected neural layers. Given the parameter 128, the first Dense layer has 128 nodes and the second node returns a logits array of length 10.
* Each node has a score that designates which of the 10 classes the current image belongs to.

### Compiling the model

* A few settings are modified during the model compile step.
  + **Loss Function**: This evaluates the model's accuracy during training. To get the model to predict more accurately, this function has to be minimised.
  + **Optimizer**: with the optimizer the model is modified or updated based on the data it sees and its loss function.
  + **Metrics**: the basis on which the model is used to monitor the training and testing steps.

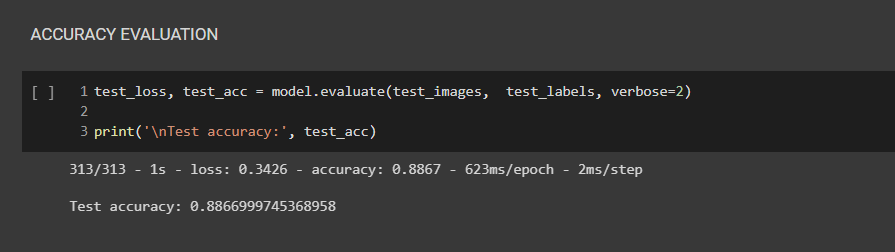
### Training the model

* To train the neural network the following must be done:
  + Feeding the training data to the model, which tis the ***train\_images*** and ***train\_labels*** arrays.
  + The model learns to link the images with the labels.
  + The predictions are made on the test\_set which is the *test\_images* array.
  + The predictions are then verified with the labels from the *test\_labels* array.
* ***Feeding the model:***
  + To begin he training, we call the ***model.fit()***method’

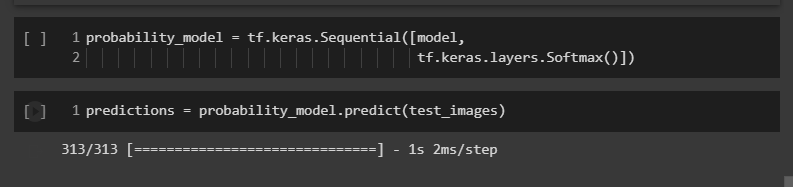


* + The loss and accuracy measures are shown as the model trains. The current model reaches an accuracy of about 0.92 (or 92%) on the training data.

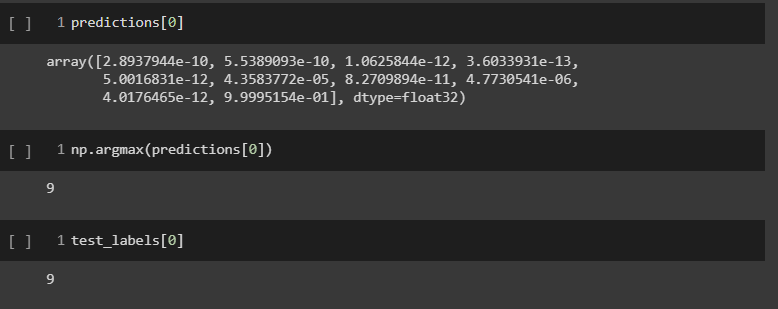
### Evaluating the accuracy

* Now to evaluate the accuracy of the model on the test dataset.
* We can clearly see that the accuracy on the test dataset is a bit less than the accuracy of the model on the training dataset.
* The gap between the testing accuracy and the training accuracy of the model represents overfitting of the model.
* ***Overfitting*** takes place when the machine learning model performs worse on new data’s input which is previously unseen in the training dataset.
* The overfitted model memorizes the noise details in the training dataset to the point where it adversely affects the model's performance on the new data absent in the training dataset.

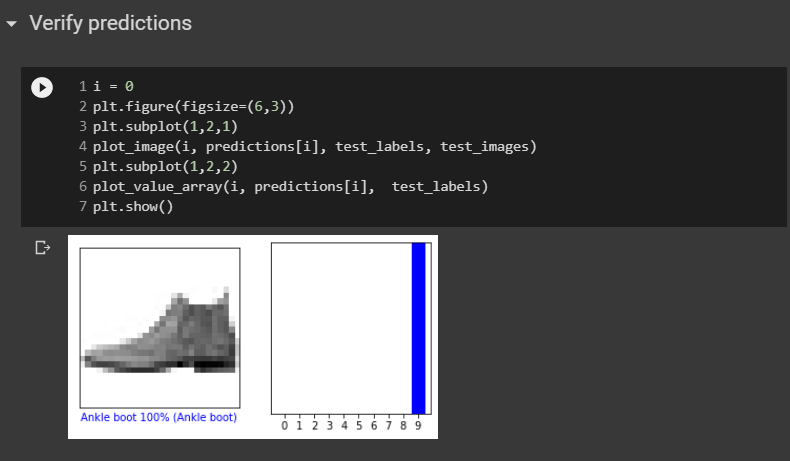
### Making the predictions

* With the trained model , the prediction can now be made on any input of images.
* With a softmax layer attached, the model’s linear outputs can be converted to probabilities.

For instance,here is the first prediction:

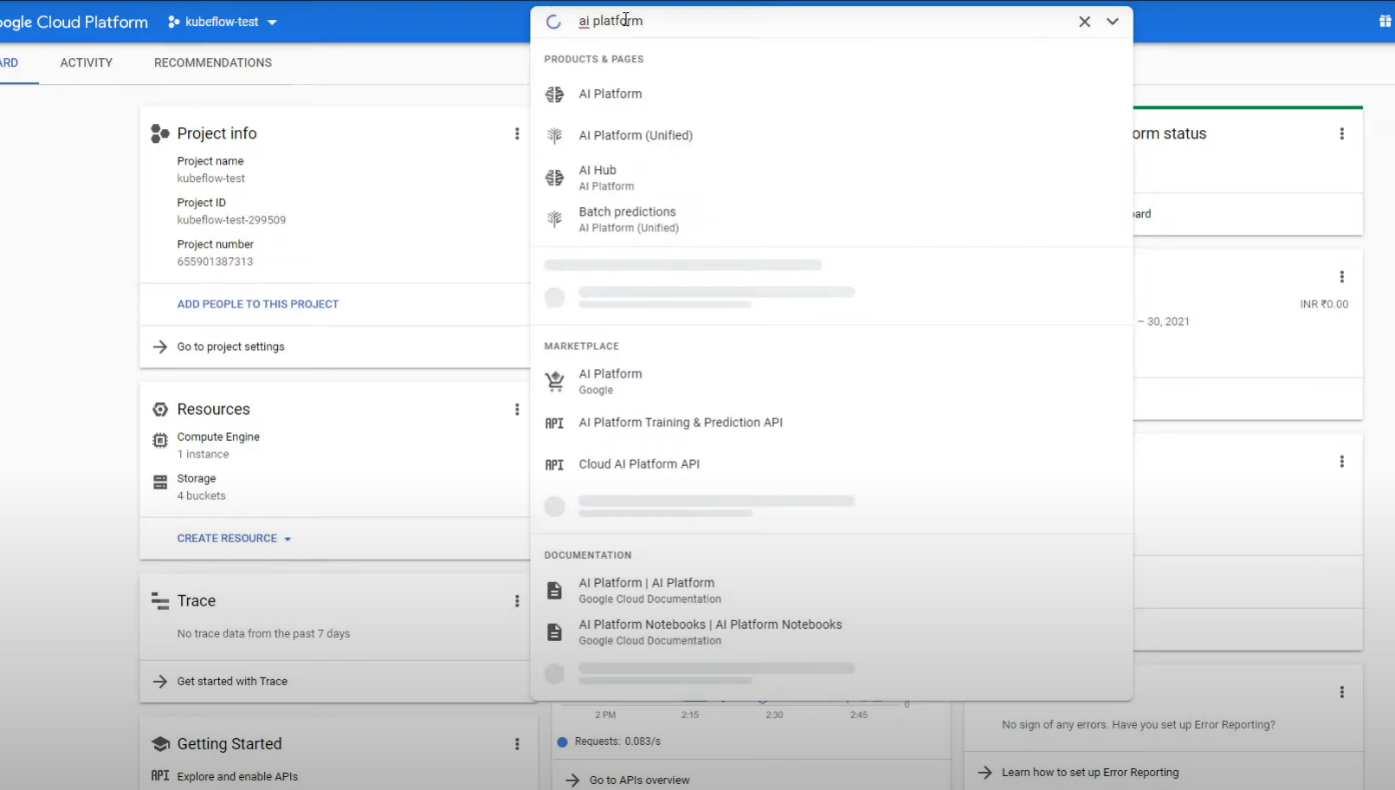


* A model prediction is an array of 10 numbers, which represent the model confidence, which corresponds to each of the 10 different articles.
* From the above result when making the prediction, we can see that the model predicts the image to be of label to be 9, which is ***class\_names[9],*** an ankle boot.
* The test label also shows the label to be 9 which shows the prediction is correct.



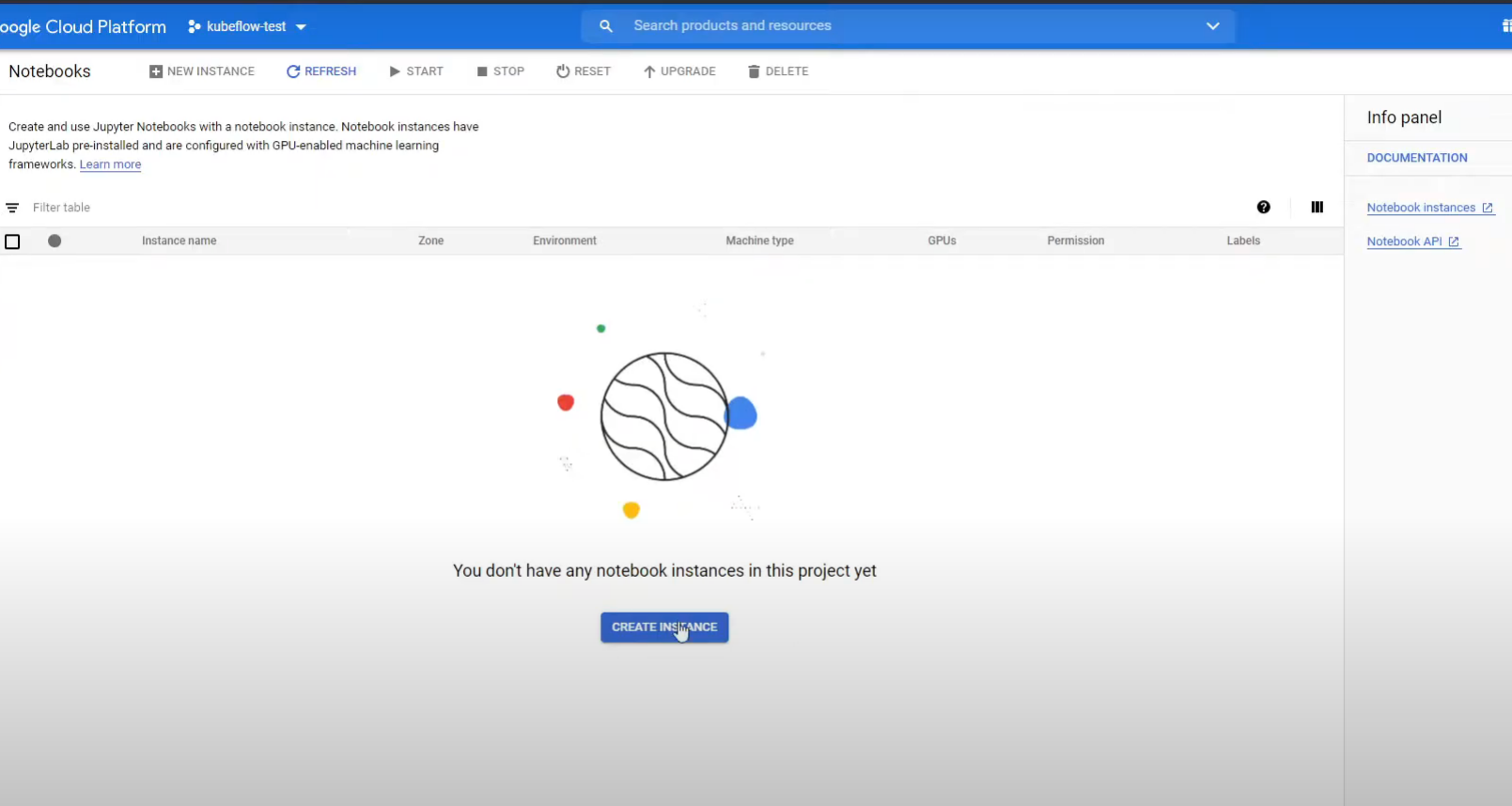
## Building the Kubeflow Pipeline

### Deploying Kubeflow

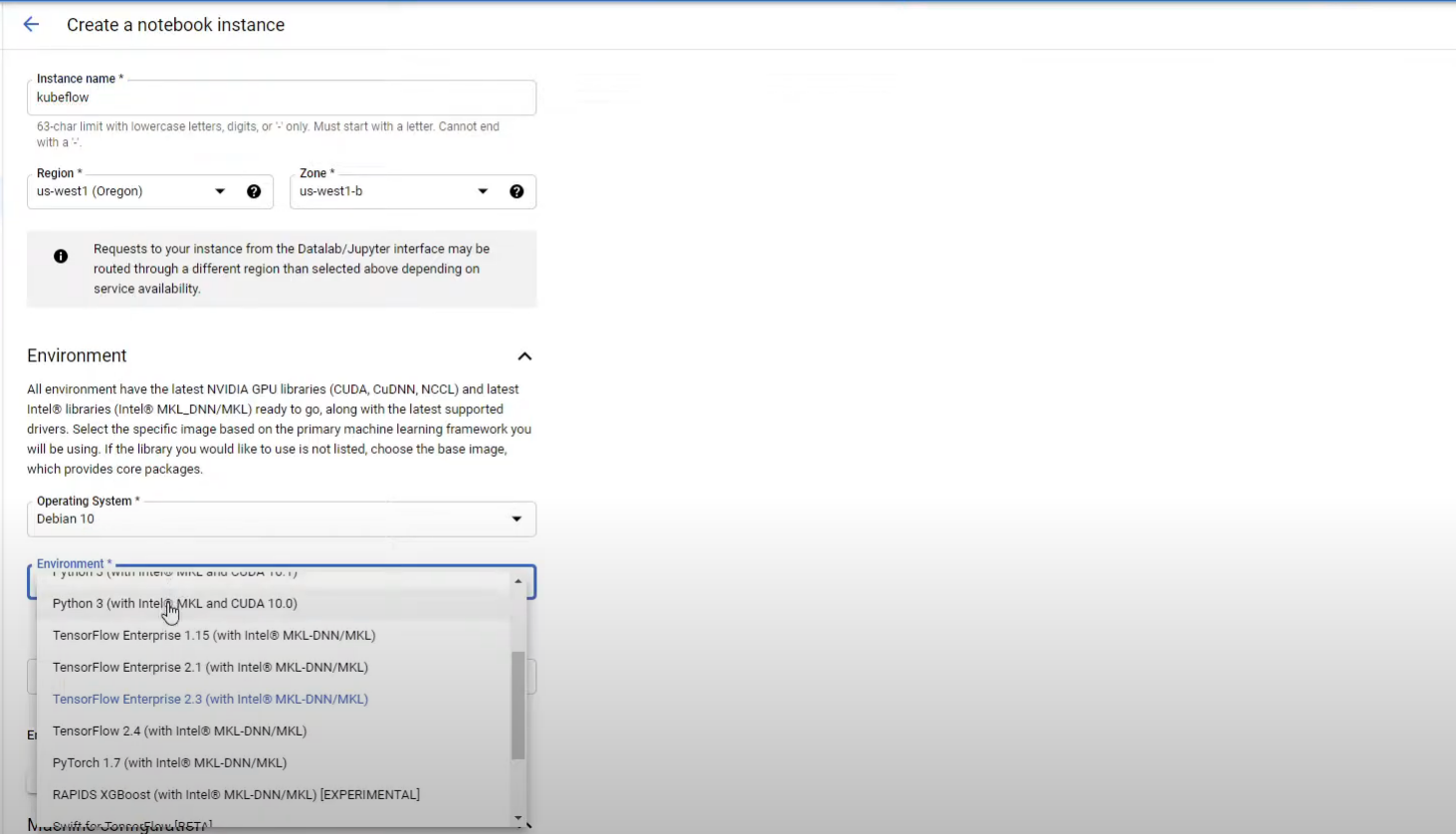
* Our objective to deploy the model is to first create a hook instance or AI Platform instance on the Google Cloud Platform
* Kubeflow package can be deployed using any of the methods mentioned above.
* Once the Kubeflow package is deployed we can access the Kubeflow dashboard.

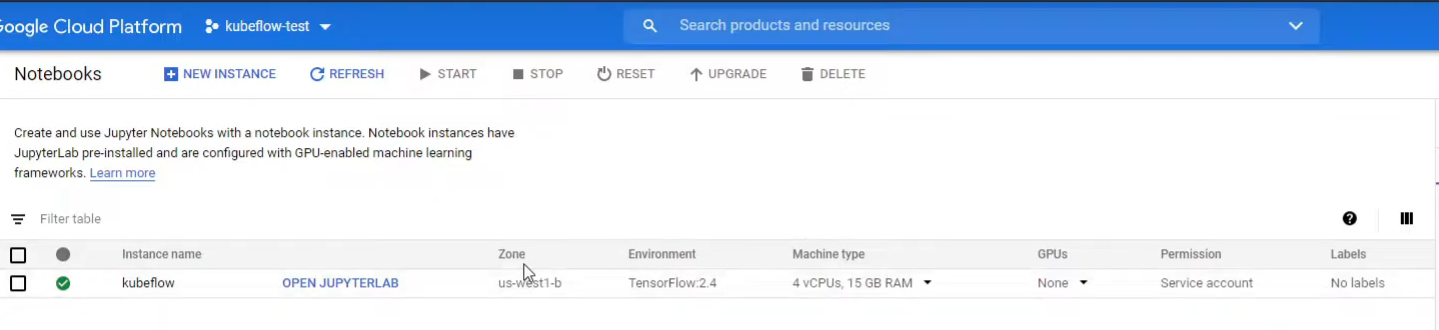
### Opening Jupyter Notebook Server

* The Jupyter Notebook server can be accessed from dashboard. In the created AI platform we create a notebook instance and select the correct required TensorFlow version
* Once the notebook server is up, open a new terminal and the notebook used to create the Fashion MNIST model can be accessed from here.



* Giving the instance name and the right Tensorflow version required.

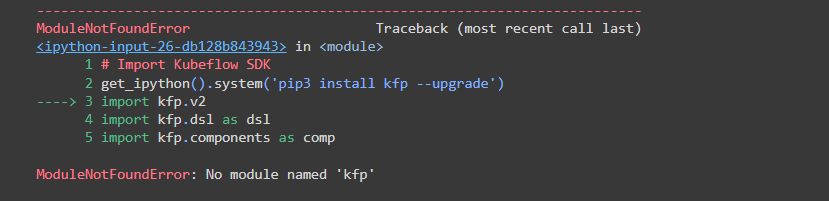




* With The notebook instance created, we import the create ML model notebook on the above JupyterLab

### Initiating the Kubeflow pipelines SDK

* Now to get started with creating the pipeline to convert the model into a running pipeline, the Kubeflow Pipelines SDK is required to be installed.
* The kernel has to be restarted after installing Kubeflow package, else we might get the below error.



### Converting to Docker containers

* The Kubeflow Python SDK allows the user to build lightweight component by the defining the python functions/methods and converting them accordingly.
* To package the written python code inside containers, the corresponding python functions must ber defined. This code could be any logical step required in the pipeline. The functions could be to train or predict the model etc.
  + The train component will train, evaluate and save the model
  + The predict component takes the model and makes the prediction on an image from the test dataset. It grabs an image from the test dataset and predicts the label for the image.

### Defining the Kubeflow pipeline

* Kubeflow uses Kubernetes resources which are generally defines using YAML templates.. The pipeline SDK allows the user to define how the code to be run without having to manually modify the YAML files.
* Kubeflow creates a compressed YAML file at compile time which defines the pipeline. These files can be reused for making the pipelines more scalable.
* A Kubeflow client has to then be initiated. The client contains client libraries for the Kubeflow Pipelines API allowing the user to further create experiments and run within thise from the Jupyter notebook
* The pipeline name and description can then be written to better be viewed and visualized on the Kubeflow Dashboard.
* The parameters required to be fed into the pipeline are then defined. For this project, the required definitions are the path for where the data will be written, the file where the model is to be stored and the integer representing the index of an image in the test dataset.

### Creating a Persistent Volume

* Persistent Volumes:
  + Containers were developed to be stateless to speed up the application launch and runtime.
  + The design tends to be problematic when the data needs to stay or persist when the container departs. The problem increases as the scale of the managing the live container deployments with tools such as Docker and Kubernetes grows.
  + *Kfp* allows in creating persistent volumes using *VolumeOp*  object.
* Persistent Volumes resolve the above issue by storing the data and not losing it in case the notebook is terminated for any reason.

1. vop = dsl.VolumeOp(

2. name=”create volume”,

3. resource\_name=”data-volume”,

4. size=”1gi”,

5. modes=dsl.VOLUME+MODE+RWM)

6.

*VolumeOp* parameters include:

* **name** : the name given for the creating the volume in the UI
* **resource\_name**: name which is referenced byu other resources
* **size**: size of the claimed volume
* **modes**: access m0odes for the volume

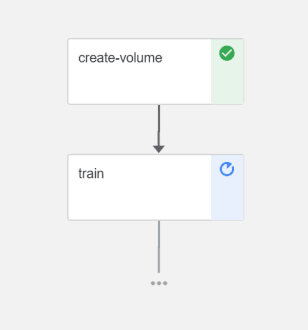
### Defining the pipeline component

### Running the pipeline

* The notebook finally, compiles the pipeline cvode and runs it within an excperiment.
* The name of the run and of the experiment is specified in the notebook and then presented in the Kubeflow dashboard.
* The pipeline can be viewed running in the Kubeflow Pipelines UI.

# Outcome

* With the pipeline created it is ready to be run. The pipeline can be viewed under the Kubeflow Pipelines dashboard.
* Each component defined in the pipeline is displayed under this tab.
* The status of each component is displayed as the path of the pipeline is updated.



* To see further details for each component, it can be viewed by clicking directly ion the component and dig into a few more tabs. The logs =generated can be viewed by clicking on the logs tab.
* Once the *echo\_result* component is done executing, the class of the image is displayed after having being predicted. Along with the class, the Confidence of the model and the actual label of the image is displayed.

# References